Enhancing the performance of frequent pattern mining with GSpan using multithreading

Jignesh Patel, Bhavesh A. Oza
Student, Assistant Professor
1Computer Science and Engineering, L.D. Engineering College, Ahmedabad, India
2Computer Department, L.D. Engineering College, Ahmedabad, India
1jigneshj204@gmail.com, 2bhavesh.oza.123@gmail.com

Abstract—The primary goal of data mining is to extract statistically significant and useful knowledge from data. The data may be any form such as text, images, and videos and so on. Graph Mining is an active area of research. Frequent pattern mining is important part of which help to discover patterns which represents relations among discrete entities. Frequent patterns are subsequences, substructures or item sets that appear in a data set with frequency which is no less than a user given threshold. Problem of frequent subgraph mining is to find frequent subgraphs over a collection of graphs. Frequent subgraph mining can be useful for providing meaningful structured information such as hot web access patterns, common protein structures and shared patterns in object recognition. It can also use for identified the attacker in in electronic payments. Another application of the frequent pattern mining is to cluster XML documents based on their common structures.

IndexTerms — Graph Mining Frequent Pattern Mining, DFS Code, Multithreading

In this paper, we propose new algorithm which is based on Gspan algorithm. In this algorithm we have used multithreading for reducing the cost for the subgraph isomorphism test. Gspan algorithm uses two techniques, DFS lexicographic order and minimum DFS code. But still the problem of finding minimum DFS code in Gspan is also NP-Complete. The Proposed algorithm is used multithreading for generating minimum DFS Codes whenever it scans the available Datasets.

2.BASIC GRAPH THEORY

2.1 Graph
A Graph G (V, E) is defined as set vertices V (nodes) which are interconnected by a set of edges E (links). Below, figure (a) shows the simple graph with three nodes and three vertices.

1.INTRODUCTION
Mining Graph is an active research area now days. Mining frequent subgraphs refers to problem of subgraph isomorphism test. In subgraph isomorphism test a mapping from nodes of graph g1 to nodes of the other graph g2 is bijective that preserves all edges and labels. Main Problem of this subgraph isomorphism test is its high computational complexity. It is also NP-Complete problem. Many algorithms have been developed for speed up this subgraph isomorphism test but no one can avoid worst case computation.

2.2 Labeled Graph
A labeled Graph can be represented as
G (V, E, Lv, Le, µ), Where V is set of Vertex ,
E ⊆ V×V is a set of edges, L_V and L_E are sets of vertex and edge labels respectively, μ is label function that defines the mapping V → L_v and E → L_E. G is (UN) directed if ∀e ∈ E, e is an (UN) ordered pair of vertexes. A path in G is a sequence of vertexes which can be ordered such that two vertexes form an edge if and only if they are consecutive in the list. G is connected, if it contains a path for every pair of vertexes in it and disconnected otherwise. G is complete if each pair of vertexes is joined by an edge and G is acyclic if it contains no cycle. Below figure (b) shows simple labeled graph with three nodes and three vertices.

2.3 Subgraph
Given two Graph G1 (V1, E1, LV1,LE1,µ1) and G2 (V2, E2, LV2,LE2,µ2), G1 is a subgraph of G2, if G1 satisfies:

i) V1 ⊆ V2, and ∀v ∈ V1, μ1(v) = μ2(v),

ii) E1 ⊆ E2, and ∀(u, v) ∈ E1, μ1 (u, v) = μ2 (u, v).

G1 is an induced subgraph of G2, if G1 furthersatisfies: ∀u, v ∈ V1, (u, v) ∈ E1 ⇔ (u, v) ∈ E2, in addition to the above conditions. G2 is also a supergraph of G1.

2.4 Graph Isomorphism
A graph G1 (V1, E1, LV1,LE1,µ1) is isomorphic to another graph G2 (V2, E2, LV2,LE2,µ2), if and only if a bijection

f:V1 → V2 exists such that:

i) ∀u ∈ V1, μ1 (u) = μ2 (f (u)),

ii) ∀(u, v) ∈ E1 ⇔ (f (u), f (v)) ∈ E2,

iii) ∀(u, v) ∈ E1, μ1 (u, v) = μ2 (f (u), f (v)).

The bijection f is an isomorphism between G1 and G2. A graph G1 is subgraph isomorphic to a graph G2, if and only if there exists a subgraph g ⊆ G2 such that G1 is isomorphic to g. In this case g is called an embedding of G1 in G2.

3. Basics of Gspan Algorithm
3.1 DFS Code
For the given graph each graph is mapped into an edge sequence called the DFS code. It is generated from the DFS Tree for the given graph.

3.2 DFS Tree
For a given graph data set DFS Tree represents the relationship between Graphs. In a DFS Code Tree each node represents a DFS Code. In DFS Code Tree at the n+1 level represents graph with n-edges.
3.3 Right Most Extension

In graph mining Right Most Extension is widely used. It is uses Depth First Search for finding the next nodes in searching the graph. In Graph extension to a node or edge path from given node or edge is called right most extension path. Right most extension can be two types : (1) Forward Extension – This extension extends a graph by adding a new node to the node of right most path (2) Backward Extension – This extension extends a graph by adding a new edge to the given edge of right most path.

4. Gspan Algorithm

Graph-Based Substructure Pattern Mining Gspan\(^{[1]}\) is a novel algorithm which discovers all frequent substructures without candidate generation as per it is uses the frequent pattern mining concept. It uses minimum DFS code concept for mapping the graph and provide the lexicographic order to generated DFS code. Gspan algorithm uses Depth first search for searching purpose. Gspan algorithm uses for discovering all available frequent substructures without any candidate generation and false positive pruning.

4.1 The Gspan algorithm

Algorithm gspan(D, minSupport, S)

1: sort labels of the vertices and edges in D by frequency;
2: remove infrequent vertices and edges;
3: relabel the remaining vertices and edges (descending);
4: S0=code of all frequent graphs with single edge;
5: sort S0 in DFS lexicographic order;
S=S0;
6: for each code s in S0 does
7: gSpan(s, D, MinSup, S);
8: D:=D-s;
9: if |D|<Minus;
10: break;

Algorithm gSpan(s, D, MinSup, S)

1: if s! =min(s), then
2: return
3: insert s into S
4: set C to { } 
5: scan D once; find every edge e such that s can be right-most
Extended to frequent s*e;
Insert s*e into C;
6: sort C in DFS lexicographic order;
7: for each s*e in C do
8: Call gSpan(s*e, D, MinSup, S);
9: return

5. Proposed Algorithm

5.1 Proposed Scheme

In Proposed algorithm gspan can be improved by the using multithreading for generating the DFS code.

Figure DFS CODES GROWTH\(^{[4]}\)

As shown in above figure for graph (a) there are five possibilities (b-f) to generate dfs code by extending the right most edges or generating new nodes. In old algorithm it generated independently instead of it may be possible to generate each dfs code by assign individual thread. So the performance of the Algorithm for the generating the DFS CODE will be increase in terms of time complexity.

5.2 Algorithm

1: sort labels of the vertices and edges in D by frequency;
2: remove infrequent vertices and edges;
3: relabel the remaining vertices and edges (descending);
4: S0=code of all frequent graphs with single edge;
5: sort S0 in DFS lexicographic order;
S=S0;
6: for each code s in S0 does
7: gSpan(s, D, MinSup, S);
8: D:=D-s;
9: if |D|<Minus;
10: break;

Algorithm gSpan(s, D, MinSup, S)

1: if s! =min(s), then
2: return
3: insert s into S
4: set C to { }
5: scan D once; find every edge e such that s can be right-
most
Provide each edge or nodes with new threads
//update with multithreading
Extended to frequent s*e;
Insert s*e into C;
6: sort C in DFS lexicographic order;
7: for each s*e in C do
8: Call gSpan(s*e, D, MinSup, S);
9: return

5.2 Dataset Information
I have used the protein data bank available on http://www.rcsb.org/pdb. It provides the information of each atom vertex entry in the graph files also contains a comment with the x, y and z coordinate for the 3D location of the atom (in angstroms).

5.4 Result Analysis
We have worked on the existing Gspan algorithm and proposed Gspan algorithm with multithreading. Following are the implementation output, analysis data and plotted result of existing and proposed algorithm.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Minimum Frequency</th>
<th>Time Without Threads</th>
<th>Time With 10 Threads</th>
<th>Time With 50 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>0.675</td>
<td>0.545</td>
<td>0.519</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0.643</td>
<td>0.569</td>
<td>0.514</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>0.599</td>
<td>0.541</td>
<td>0.51</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.563</td>
<td>0.548</td>
<td>0.518</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>0.517</td>
<td>0.548</td>
<td>0.509</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>0.518</td>
<td>0.549</td>
<td>0.538</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
<td>0.56</td>
<td>0.544</td>
<td>0.508</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>0.551</td>
<td>0.555</td>
<td>0.52</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
<td>0.645</td>
<td>0.55</td>
<td>0.547</td>
</tr>
</tbody>
</table>

Table 1 Implementation Result

Figure Graph of implementation result

6. Conclusion
In this paper, we propose a new algorithm based on gSpan using multithreading for frequent sub graph mining. In proposed algorithm we have generate the different threads for each fragment. So the choosing a new node or edges will become simpler and we will get the high speed compare to available gSpan algorithm.

References
[1] Xifeng Yan, Jiawei Han “Gspan: Graph-Based Substructure Pattern Mining” September 2002.